BANNERS: Binarized Neural Networks with Replicated Secret Sharing

Alberto Ibarrondo
IDEMIA & EURECOM
Sophia Antipolis, France
ibarrond@eurecom.fr

Hervé Chabanne
IDEMIA & Telecom Paris
Paris, France

Melek Önen
EURECOM
Sophia Antipolis, France

ABSTRACT

Binarized Neural Networks (BNN) provide efficient implementations of Convolutional Neural Networks (CNN). This makes them particularly suitable to perform fast and memory-light inference of neural networks running on resource-constrained devices. Motivated by the growing interest in CNN-based biometric recognition on potentially insecure devices, or as part of strong multi-factor authentication for sensitive applications, the protection of BNN inference on edge devices is rendered imperative. We propose a new method to perform secure inference of BNN relying on secure multiparty computation. While preceding papers offered security in a semi-honest setting for BNN or malicious security for standard CNN, our work yields security with abort against one malicious adversary for BNN by leveraging on Replicated Secret Sharing (RSS) for an honest majority with three computing parties. Experimentally, we implement BaNNeRS on top of MP-SPDZ and compare it with prior work over binarized models trained for MNIST and CIFAR10 image classification datasets. Our results attest the efficiency of BaNNeRS as a privacy-preserving inference technique.

CCS CONCEPTS

• **Security and privacy** → **Information-theoretic techniques**.

KEYWORDS

secure multiparty computation, binarized neural networks, secure inference, replicated secret sharing, privacy preserving technologies

1 INTRODUCTION

Machine Learning has become an essential tool for private and public sector alike, by virtue of the prediction capabilities of forecasting models or insights gained from recommender systems. Requiring orders of magnitude more data than classical Machine Learning, the recent progress in Deep Learning has attained models with near human capabilities to solve complex tasks like image classification [43], object detection [37] or natural language processing [10], also reaching unheard-of generation capabilities for text [10], audio [35] and image generation[50].

Reference:
Alberto Ibarrondo, Hervé Chabanne, and Melek Önen. 2020. BANNERS: Binarized Neural Networks with Replicated Secret Sharing. In *IACR eprint archive (https://eprint.iacr.org/2021/)*, 12 pages.

Making use of the deep learning toolbox comes at a non negligible cost: one needs to acquire very large amounts of structured data, considerable computational power and vast technical expertise to define and train the models. Subsequently, the expensively trained deep learning models can be used to perform inference on data not present during training. Naturally, risk arises when training or inference computation tasks are outsourced, following the trends of Cloud Computing (where the model is sent to the cloud) or Edge Computing (where the trained model is pushed to edge devices such as mobile phones or cars). In a standard setup, carrying out these processes on an outsourced enclave forces users to keep the model in plaintext to carry out mathematical operations, leading to potential model theft and exposing all intermediate computations as well as the input data and the inference result.

Moreover, there are sectors where this risk is inacceptable or even illegal due to the limitations on sharing data (GDPR in Europe, HIPAA for medical data in US). Hospitals and health specialists are deprived of the advantages of training and using models with all the available data from patients, which is proven to be very effective to tackle genome-wide association studies: associating certain genes to illnesses such as cancer for early detection and further understanding [32]. Banks, finance institutions and governments are limited to the locally available data to prevent fraud and prosecute tax evasion. Biometric Identification must rely on secure hardware or trusted parties to hold the personal data vital for their recognition models. Child Exploitative Imagery detection models [45] need training data that is in itself illegal to possess.

Under the field of advanced cryptography, several privacy preserving technologies aim to deal with these issues. *Differential Privacy* [2] provides privacy to individual elements of the training data set while keeping statistically significant features of the data set to train deep learning models, at a cost in terms of accuracy and almost no extra computation. *Fully Homomorphic Encryption* (FHE)[20] is a costly public-key encryption scheme that supports certain operations between ciphertexts (typically addition and multiplication), yielding the results of these operations when decrypting. Secure Multiparty Computation (MPC) covers a series of techniques (garbled circuits[49], secret sharing[41], or the more recent replicated secret sharing[5] and functional secret sharing[9]) that split computation of a given function across multiple distinct parties, so that each individual party remains ignorant of the global computation, and collaborate to jointly compute the result. *Functional Encryption* [7] is a computationally-expensive public-key encryption scheme

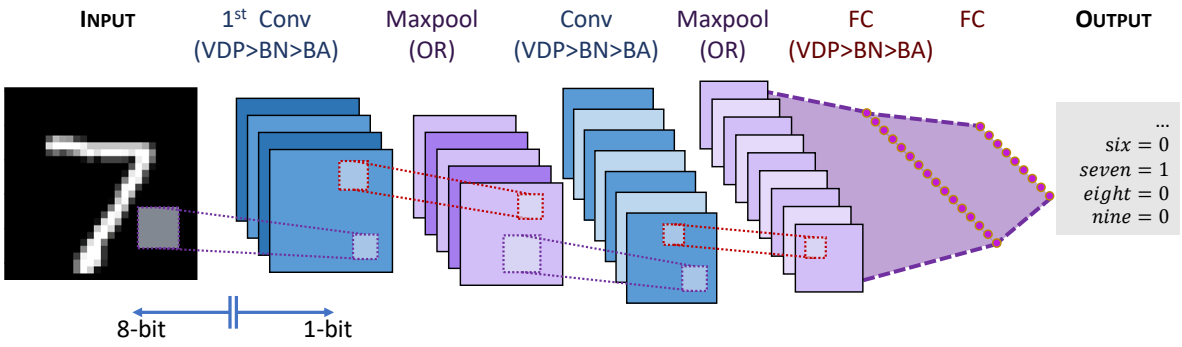

Figure 1: BNN architecture for image classification. This corresponds to the BM3 architecture for MNIST dataset in section 5.

that supports evaluation of arbitrary functions when decrypting the ciphertexts, where the decryption key holds the information about the function to be computed, and the original data can only be retrieved with the original encryption key. MPC is, at the time of this writing, among the most performant technologies providing secure outsourced computation. This work uses MPC to carry out secure inference of Neural Networks. Going beyond the Honest-But-Curious adversary model present in the majority of MPC-based secure NN inference schemes proposed so far, in this work we use a threat model whereby honest parties can abort when detecting a malicious adversary.

Motivation. From secure banking access to government services such as border control, or in general as part of strong multi-factor authentication, there is a growing interest on using Biometric Recognition for sensitive applications on potentially insecure devices [18]. Addressing the protection of biometric identification algorithms on resource-constrained devices is thus rendered imperative for industry leaders in biometric solutions [27]. Since biometric algorithms are nowadays based on modern Convolutional Neural Networks (CNN), we focus on securing the inference of these networks. A more detailed state of the art on securing CNN inference can be found in section 3. Furthermore, CNN can be binarized (constrain weights and intermediate operations to 0 and 1) in order to greatly reduce the model size and memory usage, making the resulting Binarized Neural Networks (BNN)[25] suitable to execute in edge devices such as mobile phones. BANNERS serves as the first step in this direction, implementing BNN with RSS in a suitable security model that will later be ported to the use case of biometrics.

Our contribution. Leaning on RSS, this paper proposes a new method to perform secure inference of Binarized Neural Networks, guaranteeing security with abort against one malicious adversary in a 3 party setting. The paper is outlined as follows. Section 2 covers the preliminaries, from BNN to MPC and RSS, including the security model. Section 3 builds upon those preliminaries to discuss related previous work. Section 4 presents our detailed solution, covering each and every protocol we need. Section 5 describes our implementation and experiments, closing up with conclusions and future work on section 6.

2 PRELIMINARIES

2.1 Binarized Neural Networks

BNN [25] are a subtype of Neural Networks whose weights and activations are constrained to two values $\{-1, 1\}$ (mapped into binary values $0, 1$), taking up one bit per value while sacrificing accuracy with respect to their full precision counterparts. Thanks to this limitation, up to 64 bits can be packed together in a 64-bit register, providing high parallelization on the operations in a Single Instruction Multiple Data (SIMD) fashion. This packing technique is named *Bit Slicing* [5][11], and it yields savings of up to 64 times in memory and space. Indeed, this makes BNN particularly suitable for edge devices and resource-constrained scenarios.

We implement each of the layers of a XNOR-Net[36] BNN architecture.

2.1.1 First linear layer. Linear combination of the inputs x with some weights w, there are two types of linear layers: Fully Connected (FC, also known as Dense in popular frameworks) and Convolution (Conv). FC corresponds to a matrix multiplication, whilst Conv can be turned into a matrix multiplication by applying a Toeplitz transformation on the inputs and weights. This transformation is more commonly known as `im2col` & `col2im` (more info in section 5.1 of SecureNN[46], and a nice visual explanation in slide 66 of [15]). In the end, both FC and Conv are computed as a matrix multiplication, which can be decomposed into *Vector Dot Products* (VDP). Figure 2 represents one VDP in the first layer of our BNN architecture, with 8-bit inputs and 1-bit weights.

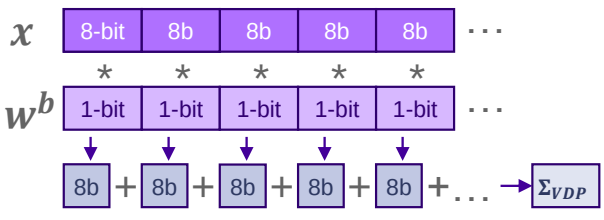

Figure 2: Diagram of a VDP in the first layer of a BNN

There is one peculiarity with the first linear layer of a BNN: Binarizing the input of the first layer would hurt accuracy much more than binarizing other layers in the network (see figure 1). Besides, the number of weights and operations in these layers tend to be relatively small. Therefore it has become standard to leave the input of this layer with higher precision (8 bits in our case).

2.1.2 Binary Activation and Batch Normalization. A *Binary Activation* (BA) is equivalent to the $Sign(x)$ function [25], and is normally applied after a linear layer. Given that the result of the VDP in linear layers is a small integer (up to $log_2(N)$ for binary VDP and $8 * log_2(N)$ for the first layer, for vectors of size N). This functionality is implemented by extracting the most significant bit (MSB).

A *Batch Normalization* (BN) operation normalizes all the inputs by subtracting β and dividing by γ, two trainable parameters. While the original batch normalization[28] includes subtracting the mean of the input batch and dividing by its standard deviation, the binarized version can be implemented by relying solely on β and γ [36][38]. Binary BN is most frequently located right before a BA. Together, a BN followed by a BA is equivalent to $sign(x - \beta/\gamma)$, instantiated as a comparison.

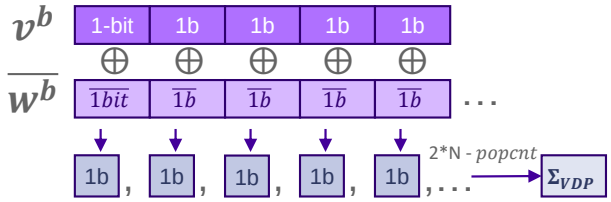

Figure 3: Diagram of a binary VDP

2.1.3 Binary linear layer. Except for the first layer, all the linear layers in a BNN have binary inputs and binary weights. Likewise, FC and Conv are turned into matrix multiplication and decomposed into a series of binary VDP. Following [36], and nicely displayed in figure 2 of XONN[38], binary VPD is equivalent to XNOR (substitute of binary multiplication) and $2 * N - popcount(x)$ (analogous to cumulative addition). Thus effectively transforming $mult\&add \rightarrow XNOR\&popcount$. Figure 3 displays the structure of an individual binary VDP.

2.1.4 Maxpool layer. A maxpool layer over binary inputs is homologous to the OR operation, as shown in figure 4

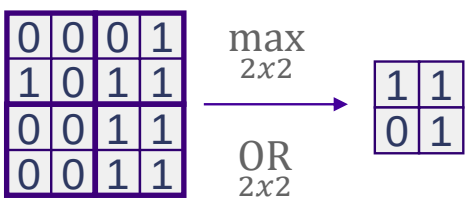

Figure 4: Equivalence between Binary *max* and boolean *OR* for a Maxpool layer

2.2 Secure Multi-Party Computation

MPC allows several mutually distrusting parties to carry out together computations on some private input, so that an adversary controlling a fraction of those parties can not learn any information beyond what is already known and permissible by the computation. In our setup, we consider a BNN model owner, an input data owner and multiple parties/servers performing the secure computation.

There are two main approaches to MPC. In *Garbled Circuits*(GC) a computing party named the *garbler* encrypts a Boolean circuit in the form of truth tables using keys from each of the parties and randomly permutes the rows. Later on, the *evaluator* collaborates to sequentially decrypt single rows of the logic gates' truth tables while the garbler remains oblivious to the information exchanged between them by using a primitive named Oblivious Transfer. The second approach, named *Secret Sharing*(SS), splits each individual data element into N shares sending one share per party, so that less than k shares reveals nothing of the input and k or more shares reconstruct the original secret without error. This approach offers cheap addition operations on local shares and communication between parties to exchange shares without ever revealing more than $k - 1$ shares to any computing party. A third technique, named GMW[22], is a binary version of SS first defined alongside other MPC primitives like *Oblivious Transfer*.

In the context of Neural Network operations, GC[49] and GMW are historically more suited for non-linear operations like comparisons, threshold-based activation functions and MaxPool, while standard (arithmetic) SS shines when used for integer addition and multiplication, which is why several previous works focused on switching between GC and SS [29][39].

On adversaries. Semihonest adversary (also known as honest but curious) defines an adversary that will follow the given computation instructions while trying to extract as much information as possible from the process. It requires passive security to overcome, making sure that data remains private, but without the need to verify the result of operations. Contrary to it, malicious adversaries (also known as dishonest) can deviate arbitrarily from the requested computation, forcing the verification of each operation to ensure correctness.

On number of parties and majorities. An honest majority comprises strictly less than half of the computing parties being corrupted by an adversary, whereas a dishonest majority involves at least half of the computing parties being potentially corrupted. Generally speaking, the complexity of MPC intensifies with the number of parties. Typically the best setup for dishonest majorities is Two Party Computation (2PC). In contrast, 3PC is particularly beneficial for an honest majority setting, since each honest party can always rely on the honesty of at least one other party. By comparing the results of the other two parties for a given computation, an honest party in this setting can cheaply detect malicious behavior and abort the computation[5][17].

On security guarantees As a general rule, stronger security is coupled with higher complexity (inferred from table 3 of [14]). We can classify the security guarantees (a gentle introduction found in [13]) of our protocols into:

- Private computation: parties cannot extract any information from the computation.
- Security with abort: the computation remains private, and if the adversary deviates, honest parties detect it and halt the computation. It does not protect against Denial of Service (DoS) attacks.
- Security with public verifiability: the computation remains private, and in case the adversary deviates, honest parties identify which party cheated and abort.
- Fully secure: the computation is ensured to yield the correct output. For Semihonest adversaries, it is equivalent to private computation.

Overall, the setting for Banners consists of an honest majority over 3PC, providing security with abort against one malicious adversary. The next sections are tailored to these choices., using notation from table 1.

Table 1: Notation for standard and replicated secret sharing

	3-out-of-3 shares (SS)	2-out-of-3 shares (RSS)
Integer	$\langle \cdot \rangle$	$\langle\!\langle \cdot \rangle\!\rangle$
Binary	$[\cdot]$	$[\![\cdot]\!]$

2.3 Secret Sharing, Replicated Secret Sharing

Formally described, Secret Sharing in a 3PC setting consists of splitting a secret integer $x \in \mathbb{Z}_K$ into randomly selected shares

$$\langle x \rangle \equiv [\langle x \rangle_0, \langle x \rangle_1, \langle x \rangle_2], \langle x \rangle_i \in \mathbb{Z}_K \tag{1}$$

so that $x = \langle x \rangle_0 + \langle x \rangle_1 + \langle x \rangle_2$, and then sending each share $\langle x \rangle_i$ to party $i \in 0, 1, 2$. Considering that you need all three shares to reconstruct x, it is also named 3-out-of-3 secret sharing. Against semihonest adversaries, parties can locally compute additions and multiplications with public constants, additions with other shared secrets, and multiplication with another shared secret at a cost of one round of communication.

Comparatively, the Replicated Secret Sharing technique (based on [5], [17]) builds upon SS, joining two SS shares into an RSS share:

$$\begin{aligned} \langle\!\langle x \rangle\!\rangle &\equiv [(\langle x \rangle_0, \langle x \rangle_1),\ (\langle x \rangle_1, \langle x \rangle_2),\ (\langle x \rangle_2, \langle x \rangle_0)] \\ &\equiv [\langle\!\langle x \rangle\!\rangle_0, \langle\!\langle x \rangle\!\rangle_1, \langle\!\langle x \rangle\!\rangle_2] \end{aligned} \tag{2}$$

and sends each RSS share $\langle\!\langle x \rangle\!\rangle_i$ to party $i \in 0, 1, 2$. Given that you only need two shares to reconstruct x, it is also designated 2-out-of-3 secret sharing. The advantage of RSS over SS is that, for the same operations described above, the scheme is secure against one malicious adversary in a 3PC honest majority setting. Instead of defining $\langle\!\langle x \rangle\!\rangle_i = (\langle x \rangle_i, \langle x \rangle_{i+1})$ as in [47], it might be convenient to define $\langle\!\langle x \rangle\!\rangle_i = (\langle x \rangle_i, \langle x \rangle_i \pm \langle x \rangle_{i+1})$ as in the original paper [5].

Below we describe the instantiation of SS and RSS for integers ($K = 2^l$ for l bits) and bits ($K = 2$) that we will use in our solution, following the notation in table 1.

We denote the next (resp. previous) party to party i in the $\{P_0, P_1, P_2\}$ triplet as $i + 1$ (resp. $i - 1$).

2.3.1 Correlated randomness. Following the techniques of [5], after a brief setup phase (where common seeds are exchanged) all parties can locally compute, using pseudorandom number generators (PRNG), correlated randomness α_i with the property $\alpha_0 + \alpha_1 + \alpha_2 = 0$ and uniformly random in $\mathbb{Z}_K$. This randomness can be used by party i to secret share a value x without communication by defining $\langle x \rangle = [\langle x \rangle_i = x + \alpha_i,\ \langle x \rangle_{i+1} = \alpha_{i+1},\ \langle x \rangle_{i-1} = \alpha_{i-1}]$.

2.3.2 Integer sharing. Also known as arithmetic sharing, sharing a single integer $x \in \mathbb{Z}_{2^l}$ requires first to split x into random arithmetic shares by using randomness $\langle r \rangle_i$ uniformly random $\in \mathbb{Z}_{2^l}$ so that $r = 0 = \langle r \rangle_0 + \langle r \rangle_1 + \langle r \rangle_2$. These values are used to conceal x: $\langle x \rangle_i = x - \langle r \rangle_i$, and it naturally holds that $x = \langle x \rangle_0 + \langle x \rangle_1 + \langle x \rangle_2$. Note that the equation $\langle r \rangle_i = \langle x \rangle_{i+1} + \langle x \rangle_{i-1}$ also holds true. Following an RSS scheme, each party i receives $\langle\!\langle x \rangle\!\rangle_i = (\langle x \rangle_i, \langle x \rangle_i + \langle x \rangle_{i+1}) = [(\langle x \rangle_i, \langle r \rangle_{i-1})$ when sharing an integer secret. Addition of two integer shared secrets can be computed locally on the parties[5], and multiplication between two integer shared secrets requires one round of communication with 2 integers sent per party[5][17].

2.3.3 Binary sharing. Similarly, sharing a single bit $w \in \mathbb{Z}_2$ requires first to split w into random bit shares by using some correlated randomness $[s]_i$ uniformly random $\in \mathbb{Z}_2$ so that $s = 0 = [s]_0 \oplus [s]_1 \oplus [s]_2$. These random values are used to conceal w: $[w]_i = w \oplus [s]_i$, and it naturally holds that $w = [w]_0 \oplus [w]_1 \oplus [w]_2$. Note that the equation $[s]_i = [w]_{i+1} \oplus [w]_{i-1}$ also holds true. Following an RSS scheme, each party i receives $[\![x]\!]_i = (\langle x \rangle_i, \langle x \rangle_{i-1})$ when sharing a binary secret. Analogous to integer sharing, XOR of two binary shared secrets can be computed locally on the parties[5], whereas AND between two binary shared secrets requires one round of communication with 2 bits sent per party[5][17].

3 PREVIOUS WORK

There are several publications that serve as foundations for BaNNeRS. The original definition of RSS is depicted in [5], with [17] adapting it to the fully malicious case. ABY3 [34] was one of the first to use RSS to secure deep neural network inference, with FALCON [47] being one of the most recent and most performant approaches. Banners is inspired by certain protocols and techniques from them.

XONN [38] is the most notorious prior work addressing the subfield of secure BNN inference with MPC, relying on Garbled Circuits in a 2PC setting to secure a custom trained XNOR-Net model [36]. Consequently, we rely on the results of XONN to compare with the results of our experiments in section 5. SOTERIA [3] generalizes the Neural Architecture Search of XONN, also addressing BNN inference with GC constructions. Note that, in both cases, the security model is that of a semihonest adversary. In contrast, our work yields security against one malicious adversary. To the best of our knowledge, this is the first work tackling maliciously secure BNN inference.

In the broader field of privacy preserving Neural Network inference, there has been a plethora of works in the recent years. A good up-to-date summary can be found in Table 1 of FALCON [47]. FHE was the foundation for the seminal CryptoNets[21] and subsequent works improved it like [12] and [24], [8] for discretized networks or [26] covering BN support for FHE. A different line of works focused on efficient MPC implementations relying on various techniques,

such as Cryptflow[31] and QuantizedNN[14] or hybrids using both FHE and MPC such as Gazelle[29] or Chameleon [39].

4 OUR CONTRIBUTION

BaNNeRS makes use of RSS to protect each of the layers described in section 2 for secure BNN inference on a 3PC (parties P_0, P_1, P_2) honest majority setting. We use both binary sharing and integer/arithmetic sharing as described in [5], similarly to [34] and [47].

4.1 Input data

The input data consists of a vector $x = [x_0, x_1, x_2, \ldots, x_{N-1}], x_i \in \mathbb{Z}_{2^k}$ of N integers, while the model data consists of multiple vectors of 1-bit weights $w = [w_0, w_1, w_2, \ldots, w_{k-1}], w_j \in \mathbb{Z}_2$ (being k the number of neurons at a given layer, k=N for the first layer), that can also be represented as $y = [y_0, y_1, y_2, \ldots, y_{k-1}], y_j \in \{-1, 1\}$ by the bijective mapping $y_j \leftrightarrow w_j : \{-1 \leftrightarrow 0, +1 \leftrightarrow 1\}$. We define $v = [v_0, v_1, v_2, \ldots, v_{k-1}], v_j \in \mathbb{Z}_2$ as the vector of bits used as input to an arbitrary hidden layer. The model data also includes the BN parameters γ and β, as well as the entire architecture of the model. Note that BANNERS requires the architecture (number of layers, type and configuration of each layer) to be publicly shared with all the computing parties. Therefore, we do not protect against model inversion [16] or model retrieval attacks[44], as it is orthogonal to our purposes.

The protocol makes use of a secure transfer of shares from the data holders to the three computing parties/servers relying on standard secure communication protocols. Input x and all the model parameters are shared with the parties using RSS.

4.1.1 On the size/format of x_j and y_j. Typically, the input of a CNN is an image whose values have been normalized (between 0 and 1), thus requiring float point arithmetic with sufficient decimals to maintain the accuracy of the first layer. However, the original rectangular image is made of RGB pixels taking integer values between 0 and 255 (in a 8-bit color map). Knowing this, we remove the normalization from the data preprocessing, relying on the first Batch Normalization layer to accomplish such task. The input values are set to 8-bits, and the shares of the inputs can also be set to 8 bits, minimizing the size of the communication while preserving security: $x_j \in \mathbb{Z}_{2^8}$. By additionally changing the input domain from $[0, 255]$ to $[-128, 127]$ we would be centering it on 0 while keeping the input distribution intact. We can interpret this as a scale shifting, which is translated implementation-wise into changing from unsigned integers to signed integers without modifying the values, all while using a fixed-point representation of signed (2s complement) integers in 8-bits. This proves useful when operating with the first layer weights. The first layer weights y_j take the mathematical values $-1, +1$ in the operation. While in the Binary layers we would map the y_j weights into bit values $w_j \in 0, 1$ as a result of the $mult\&add \rightarrow XNOR\&popcount$ transformation (see 2.1.3), in the first layer we are interested on keeping their mathematical representation to operate normally. We format them as 8-bit signed values, compressing them during communication into single bits $y_j \rightarrow w_j : -1 \rightarrow 0, +1 \rightarrow 1$ to reduce 8x the amount of communication (and reconstructing them upon reception $w_j \rightarrow y_j$). y_j is shared among parties using binary RSS on the bits w_j. Thanks to the bijective mapping, we preserve the same security properties present in binary RSS.

4.2 First layer VDP

To be consistent with previous work, we reuse notation from XONN [38]. XONN's linear operation in the first layer is defined as:

$$\rho_{XONN} = f(\mathrm{x}, \mathrm{w}) = \sum_{j=1}^{N} x_j * (-1)^{\bar{w}_j} = \sum_{j=1}^{N} x_j * y_j \tag{3}$$

This operation is carried out in BANNERS with local arithmetic multiplication further reconstruction of the RSS shares, ending with the local cumulative addition.

$$\sum_{j=1}^{N} \langle\!\langle x_j \rangle\!\rangle * (-1)^{[\![w_j]\!]} \xrightarrow[\text{mult.}]{\text{local}} \sum_{j=1}^{N} \langle z_j \rangle \xrightarrow[\text{comm.}]{\text{1 round}} \sum_{j=1}^{N} \langle\!\langle z_j \rangle\!\rangle \xrightarrow[\text{cumm. add}]{\text{local}} \Sigma_{VDP}(\rho_{XONN}) \tag{4}$$

4.2.1 For each individual multiplication. $z_j = x_j * y_j$

$$\begin{aligned} z = x * y + 0 &= (\langle x \rangle_0 + \langle x \rangle_1 + \langle x \rangle_2) + (\langle y \rangle_0 + \langle y \rangle_1 + \langle y \rangle_2) \\ &= [(\langle x \rangle_0 + \langle x \rangle_1) * \langle y \rangle_0 + \langle x \rangle_0 * \langle y \rangle_1] + \\ &\quad [(\langle x \rangle_1 + \langle x \rangle_2) * \langle y \rangle_1 + \langle x \rangle_1 * \langle y \rangle_2] + \\ &\quad [(\langle x \rangle_2 + \langle x \rangle_0) * \langle y \rangle_2 + \langle x \rangle_2 * \langle y \rangle_0] = \\ &\langle r \rangle_2 * \langle y \rangle_0 + \langle x \rangle_0 * \langle y \rangle_1 \Rightarrow \text{Locally computed in } P_0 \text{ as } \langle z \rangle_0 \\ + \ &\langle r \rangle_0 * \langle y \rangle_1 + \langle x \rangle_1 * \langle y \rangle_2 \Rightarrow \text{Locally computed in } P_1 \text{ as } \langle z \rangle_1 \\ + \ &\langle r \rangle_1 * \langle y \rangle_2 + \langle x \rangle_2 * \langle y \rangle_0 \Rightarrow \text{Locally computed in } P_2 \text{ as } \langle z \rangle_2 \end{aligned} \tag{5}$$

4.2.2 For the cumulative addition. In order to avoid overflow in the cumulative addition we need $log_2 N$ extra bits. We cast z_j from 8-bit to either 16-bit or 32-bit (depending on the size of the VDP) and perform local addition including common randomness to hide the result from other parties:

$$\begin{aligned} \rho = \sum_{j=1}^{N} \langle\!\langle z_j \rangle\!\rangle &= \sum_{j=1}^{N} \langle\!\langle z_j \rangle\!\rangle_0 + \sum_{j=1}^{N} \langle\!\langle z_j \rangle\!\rangle_1 + \sum_{j=1}^{N} \langle\!\langle z_j \rangle\!\rangle_2 + \alpha_0 + \alpha_1 + \alpha_2 = \\ &\sum_{j=1}^{N} \langle\!\langle z_j \rangle\!\rangle_0 + \alpha_0 \Rightarrow \text{Locally computed in } P_0 \text{ as } \langle\!\langle \rho \rangle\!\rangle_0 \\ &\sum_{j=1}^{N} \langle\!\langle z_j \rangle\!\rangle_1 + \alpha_1 \Rightarrow \text{Locally computed in } P_1 \text{ as } \langle\!\langle \rho \rangle\!\rangle_1 \\ &\sum_{j=1}^{N} \langle\!\langle z_j \rangle\!\rangle_2 + \alpha_2 \Rightarrow \text{Locally computed in } P_2 \text{ as } \langle\!\langle \rho \rangle\!\rangle_2 \end{aligned} \tag{6}$$

As a result we obtain 2-out-of-3 shares of ρ. We describe the entire computation in algorithm 1.

4.3 BN + BA as secure comparison

Based on [36], we make use of the transformation of BN + BA into $sign(x - \rho - \beta/\gamma)$, and while the subtraction $\rho - \beta/\gamma$ can be performed locally using shares of $\langle\!\langle \beta/\gamma \rangle\!\rangle$ gotten as part of the input

Algorithm 1 Integer-binary VPD:

Input: P_0, P_1, P_2 hold integer shares $\langle\langle x_j \rangle\rangle$ and $\langle\langle y_j \rangle\rangle$ in $\mathbb{Z}_{2^l}$
Correlated randomness: P_0, P_1, P_2 hold shares of zeroed value $\langle\langle \alpha_j \rangle\rangle$.
Output: All parties get integer shares of $\langle\langle \Sigma_{VDP} \rangle\rangle$.
Note: All shares are over $\mathbb{Z}_{2^l}$, with l large enough to avoid overflow (upper bound $\log_2(N) + 8$, based on layer size).

$\langle z_j \rangle = \langle\langle x_j \rangle\rangle * \langle\langle y_j \rangle\rangle$
P_i sends: $\langle z_j \rangle_i \rightarrow P_{i-1}$, and $\langle z_j \rangle_{i+1} \rightarrow P_{i+1}$ to verify result.
if $\langle z_j \rangle_{i+1} \neq \langle z_j \rangle_{i-1}$ **then**
  abort
else
  Reconstruct $\langle\langle z_j \rangle\rangle$
end if
$\langle\langle \Sigma_{VDP} \rangle\rangle_i = \sum_{j=1}^{N} \langle\langle z_j \rangle\rangle_i + \langle\langle \alpha_j \rangle\rangle$
return Shares of $\langle\langle \Sigma_{VDP} \rangle\rangle \in \mathbb{Z}_{2^l}$

Algorithm 2 Binary BN + BA:

Input: P_0, P_1, P_2 hold arithmetic shares of $[\![x]\!]$ in $\mathbb{Z}_{2^l}$.
Correlated randomness: P_0, P_1, P_2 hold shares of a random bit in two rings $[\![\beta]\!]2$ and $[\![\beta]\!]p$ and shares of a random, secret integer $m \in \mathbb{Z}_p^*$.
Output: All parties get shares of the bit $(x \geq 0) \in \mathbb{Z}_2$.
Note: Arithmetic shares are over $\mathbb{Z}_p$ after conversion.

$\langle\langle z \rangle\rangle = \langle\langle x \rangle\rangle - \langle\langle \beta/\gamma \rangle\rangle$
arith2bitdecomp: (from [47]) $\langle\langle z \rangle\rangle \rightarrow$ shares of bits of z, $[\![z_i]\!], i \in 1, \ldots, l$
for $i = \{\ell - 1, \ell - 2, \ldots, 0\}$ **do**
  Compute shares of $c[i] = (-1)^{\beta} z[i] + 1 + \sum_{k=i+1}^{\ell} z[k]$
end for
Compute and reveal $d := [\![m]\!]p \cdot \prod_{i=0}^{\ell-1} c[i] \pmod p$
Let $\beta' = 1$ if $(d \neq 0)$ and 0 otherwise.
return Shares of $[\![\beta' \oplus \beta]\!] \in \mathbb{Z}_2$

data, we still need a secure way to perform $q = sign(n)$. Following the $mult\&add \rightarrow XNOR\&popcount$ transformation (and its corresponding mapping $y_j \rightarrow w_j$), the $sign(n)$ function turns into $H(n)$, the Heaviside[1] function a.k.a. step function:

$$q = Heaviside(n) = H(n) = \begin{cases} 0 & \text{if } n < 0 \\ 1 & \text{if } n \geq 0 \end{cases} \tag{7}$$

As seen in previous work [46], this is equivalent to extract and negate the MSB in our fixed-point arithmetic representation. Indeed this is a step required to compute ReLU in FALCON[47] and SecureNN[46], which makes our activation function cheaper than standard ReLU.

Together, they turn into a comparison between input x and β/γ, implemented by extracting the MSB (the sign) of $x - \beta/\gamma$. We rely on FALCON's PrivateCompare (Algorithm 1 in [47]), simplifying it further by setting $r = 0$, described in algorithm 2. Since this algorithm requires shares of bits of x, we reuse the same constructions used in FALCON to switch from arithmetic shares $\langle\langle x \rangle\rangle_i \in \mathbb{Z}_{2^l}$ generated by linear layers to shares of bits of x in Z_p, with $p = 37$.

Note that, contrary to FALCON, we can directly benefit from the binary sharing returned by the private compare algorithm, since it will serve as input to subsequent binarized layers without requiring a reconversion to $\mathbb{Z}_{2^l}$.

4.4 Binary VDP

Vectorized XNOR ($t = v \oplus \bar{w}$) in a RSS setting is computed locally based on local XOR ([5] section 2.1) and local negation of 1 out of the 3 shares. Popcount is translated into cumulative addition by converting binary shares into arithmetic shares using the protocol in section 5.4.2 of ABY3 [34](simplified by setting $a = 1$):

[1]The Heaviside function is equivalent to the derivative of relu $dReLU = \frac{\partial max(0,x)}{\partial x}$. The only difference is that H(t) is defined for $\mathbb{Z}$ only, while dReLU is defined for $\mathbb{R}$

$$\sum_{j=1}^{N} [\![v_j]\!] \oplus [\![\bar{w_j}]\!] \xrightarrow[\text{XOR, NOT}]{\text{local}} \sum_{j=1}^{N} [\![t_j]\!] \xrightarrow[\text{comm.}]{\text{2 rounds}} \sum_{j=1}^{N} \langle\langle t_j \rangle\rangle \xrightarrow[\text{cumm. add}]{\text{local}} \Sigma_{VDP}(\rho_{XONN}) \tag{8}$$

With the binary input vector v, and the weights vector w, we implement their VDP using XONN's $mult\&add \rightarrow XNOR\&popcount$ transformation:

$$\sum_{j=1}^{N} [\![v_j]\!] \oplus [\![\bar{w_j}]\!] \xrightarrow[\text{XOR, NOT}]{\text{local}} \sum_{j=1}^{N} [\![t_j]\!] \xrightarrow[\text{comm.}]{\text{2 rounds}} \sum_{j=1}^{N} \langle\langle t_j \rangle\rangle \xrightarrow[\text{cumm. add}]{\text{local}} \Sigma_{VDP}(\rho_{XONN}) \tag{9}$$

4.4.1 XNOR. Starting with 2-out-of-3 shares of a vector of bits $[\![v]\!]$, and similar shares of binary weights $[\![w]\!]$, we use local evaluation of XOR from [5] to implement XOR, where $r = [r]_0 \oplus [r]_1 \oplus [r]_2$ is the correlated randomness of v and $s = [s]_0 \oplus [s]_1 \oplus [s]_2$ is the correlated randomness of w; and $r = s = 0$. Note that, using the binary sharing proposed above, party P_i holds w_i, w_{i+1}, and thus holds $s_{i-1} = w_i \oplus w_{i+1}$; respectively for v_i, v_{i+1} and r_{i-1}

$$\begin{aligned} &[\![t]\!] = [\![v \oplus w, r \oplus s]\!] = \\ &[(\,[v]_0 \oplus [v]_1 \oplus [v]_2) \oplus ([w]_0 \oplus [w]_1 \oplus ([w]_2)), \\ &\;(\,[r]_0 \oplus [r]_1 \oplus [r]_2) \oplus ([s]_0 \oplus [s]_1 \oplus ([s]_2)] \rightarrow \\ &\quad [w]_0 \oplus [v]_0, [r]_2 \oplus [s]_2 \Rightarrow \text{Locally computed in } P_0 \text{ as } [\![t]\!]_0 \\ &\quad [w]_1 \oplus [v]_1, [r]_0 \oplus [s]_0 \Rightarrow \text{Locally computed in } P_1 \text{ as } [\![t]\!]_1 \\ &\quad [w]_2 \oplus [v]_2, [r]_1 \oplus [s]_1 \Rightarrow \text{Locally computed in } P_2 \text{ as } [\![t]\!]_2 \end{aligned} \tag{10}$$

4.4.2 Popcount. The equivalent of cumulative addition for integers, *popcount* (or hamming weight) adds all the bits set to 1. To perform this cumulative addition, standard Garbled Circuits require an entire

tree of ripple carry adders (RCA)[48], as it is the case in XONN[38]. This renders the computation quite expensive, seeing how each RCA requires at least one AND operation (1 round of communication each, $log_2 N$ rounds in total). Instead, based on section 5.4.2 of ABY3 [34] we convert the binary shares into integer shares at a cost of 2 multiplications and then perform local cumulative addition over the resulting integer shares, just like in the first layer.

The conversion happens as follows:

$$\sum_{j=1}^{N} [\![t_j]\!] \xrightarrow[\text{comm.}]{\text{2 rounds}} \sum_{j=1}^{N} \langle\!\langle t_j \rangle\!\rangle \xrightarrow[\text{cumm. add}]{\text{local}} \Sigma_{VDP}(\rho_{XONN}) \tag{11}$$

The entire binary linear layer would look like this:

$$\begin{aligned}\langle\!\langle b \rangle\!\rangle =& 2 * N - \sum_{j=1}^{N} [\![v_j]\!] \oplus [\![\bar{w_j}]\!] \xrightarrow[\text{XOR, NOT}]{\text{local}} 2 * N - \sum_{j=1}^{N} [\![t_j]\!] \xrightarrow[\text{comm.}]{\text{2 rounds}} \\ & 2 * N - \sum_{j=1}^{N} \langle\!\langle t_j \rangle\!\rangle \xrightarrow[\text{cumm. add}]{\text{local}} 2 * N - \Sigma_{VDP}\end{aligned} \tag{12}$$

The actual output of the binary VDP is $2 * \Sigma_{VDP} - N$, as shown in figure 2 of XONN[38]. The complete Binary VDP is detailed in algorithm 3.

Algorithm 3 Binary VDP:

Input: P_0, P_1, P_2 hold binary shares of $[\![x_j]\!]$ in a given window spanning $(1 \dots j \dots N)$.

Correlated randomness: P_0, P_1, P_2 hold integer shares of zeroed values $\langle\!\langle a_j \rangle\!\rangle, \langle\!\langle b_j \rangle\!\rangle, \langle\!\langle c_j \rangle\!\rangle, \langle\!\langle \alpha_j \rangle\!\rangle$.

Output: All parties get integer shares of Res_{VDP}.

Note: Shares over $\mathbb{Z}_{2^l}$ are defined with l large enough to avoid overflow (upper bound $\log_2(N)$, based on binary layer size). Arithmetic multiplications in steps 6 and 7 also include the **abort** mechanism from algorithm 1.

1: $[\![t_j]\!] = [\![v_j]\!] \oplus [\![\bar{w_j}]\!]$
2: **bin2arith** $[\![t_j]\!] \rightarrow \langle\!\langle t_j \rangle\!\rangle$
3: P_0: $\langle\!\langle t_j \rangle\!\rangle_{b0} = [\![t_j]\!]_0 + \langle\!\langle a_j \rangle\!\rangle$
4: P_1: $\langle\!\langle t_j \rangle\!\rangle_{b1} = [\![t_j]\!]_1 + \langle\!\langle b_j \rangle\!\rangle$
5: P_2: $\langle\!\langle t_j \rangle\!\rangle_{b2} = [\![t_j]\!]_2 + \langle\!\langle c_j \rangle\!\rangle$
6: $\langle\!\langle d_j \rangle\!\rangle = \langle\!\langle t_j \rangle\!\rangle_{b0} + \langle\!\langle t_j \rangle\!\rangle_{b1} - 2 * \langle\!\langle t_j \rangle\!\rangle_{b0} * \langle\!\langle t_j \rangle\!\rangle_{b1}$
7: $\langle\!\langle t_j \rangle\!\rangle = \langle\!\langle t_j \rangle\!\rangle_{b2} + \langle\!\langle d_j \rangle\!\rangle - 2 * \langle\!\langle t_j \rangle\!\rangle_{b2} * \langle\!\langle d_j \rangle\!\rangle$
8: $\langle\!\langle \Sigma_{VDP} \rangle\!\rangle = \sum_{j=1}^{N} \langle\!\langle t_j \rangle\!\rangle + \langle\!\langle \alpha_j \rangle\!\rangle$
9: $\langle\!\langle Res_{VDP} \rangle\!\rangle = 2 * N - \langle\!\langle \Sigma_{VDP} \rangle\!\rangle$
10: **return** Shares of $\langle\!\langle Res_{VDP} \rangle\!\rangle \in \mathbb{Z}_{2^l}$

4.5 Max pooling

Max pooling requires computing the OR function over the values in the sliding window. However, [5] only defines NOT, XOR and AND as operations in the binary sharing domain. In order to compute OR, we reformulate OR with the available gates using NAND logic and decomposing: $OR(a, b) = NOT(AND(NOT(a), NOT(b)))$. We can now formulate the Max operation that composes a Maxpool layer:

$$\begin{aligned}m =& max_{\text{window } q}(x) = x_{q_1} \; OR \; x_{q_2} \; OR \; \cdots = \\ & not(not(x_{q_1}) \; AND \; not(x_{q_2}) AND \; \dots) \equiv \overline{\overline{x_{q_1}} \; \& \; \overline{x_{q_2}} \; \& \; \dots}\end{aligned} \tag{13}$$

As such, the Binary Maxpool layer requires as many multiplications as the number of elements in the sliding window, with 4 being a typical value. This implies one communication round per multiplication. The full layer is described in algorithm 4.

Algorithm 4 MaxPool:

Input: P_0, P_1, P_2 hold binary shares $[\![x_j]\!]$ over a window of size $1 \dots j \dots N$

Correlated randomness: P_0, P_1, P_2 hold binary shares of zeroed bits $[\![a_j]\!]$.

Output: All parties get binary shares of $[\![m]\!]_{maxpool}$.

Note: $\&(AND)$ operation is performed following [5], with **abort** conditions similar to those in algorithm 1, but applied in $\mathbb{Z}_2$. $\overline{b}(NOT)$ is performed locally negating the binary shares in P_0.

1: $[\![m]\!] = \overline{[\![x_0]\!]}$
2: **for** $i = \{2, \dots j, \dots, N\}$ **do**
3: $[\![m]\!] = [\![m]\!] \; \& \; \overline{[\![x_j]\!]}$
4: **end for**
5: $[\![m]\!] = \overline{[\![m]\!]}$
6: **return** Shares of $[\![m]\!] \in \mathbb{Z}_2$

5 EXPERIMENTS

5.1 Implementation

We implemented BANNERS on top of the versatile library MP-SPDZ [30]. As part of the implementation we created our own data management functions (`im2col`[15], `col2im`, `padding`, `flatten`), while relying on existing functionalities in MP-SPDZ to handle the MPC session and its low level operations. We report the total communication and the total online processing time, purposely leaving out offline processing.

We used Larq[19], a high level BNN framework built as an extension to Tensorflow[1], to define and train our own BNN models for image classification over the MNIST and CIFAR10 datasets. We relied on recommendations from [6] to define our custom BNN architectures, and took notions from [4] and used the Bop optimizer[23] to speed up training. To compare precisely with XONN, we applied early stopping once the desired accuracy is reached (the accuracy reported in [38] for each model). The deviation in accuracy with respect to XONN models is of 0.2%.

5.2 Comparison with XONN

As the most significant prior work on secure BNN inference, we chose to compare BANNERS with XONN[38]. In order to do so, we trained the BNNs shown in table 2, whose architectures are directly taken from XONN([38], Appendix A.2). Since XONN defines a scaling parameter `s` that increases the number of feature maps in a given BNN, we trained models for s in $(1, 1.5, 2, 3, 4)$ to compare ourselves with tables 11 and 12 from [38].

Table 2: BNN architectures trained for comparison with XONN

Arch.	Previous Papers	Description	Dataset
BM1	XONN[38], MiniONN[33]	3FC	MNIST
BM2	XONN[38], CryptoNets[21], MiniONN[33], Chameleon[39]	1 CONV, 2 FC	MNIST
BM3	XONN[38], MiniONN[33]	2CONV,2MP,2FC	MNIST
BC1	XONN[38], Chameleon[39], Gazelle[29]	7CONV,2MP,1FC	CIFAR10
BC2	XONN[38], Fitnet[40]	9CONV,3MP,1FC	CIFAR10
BC3	XONN[38], Fitnet[40]	9CONV,3MP,1FC	CIFAR10
BC4	XONN[38], Fitnet[40]	11CONV,3MP,1FC	CIFAR10
BC5	XONN[38], Fitnet[40]	17CONV,3MP,1FC	CIFAR10
BC6	XONN[38], VGG16[42]	13CONV,5MP,1FC	CIFAR10

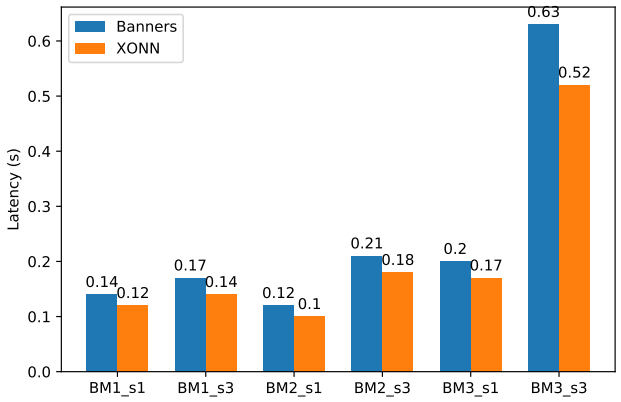

Figure 5: Comparison in latency for MNIST BNN models

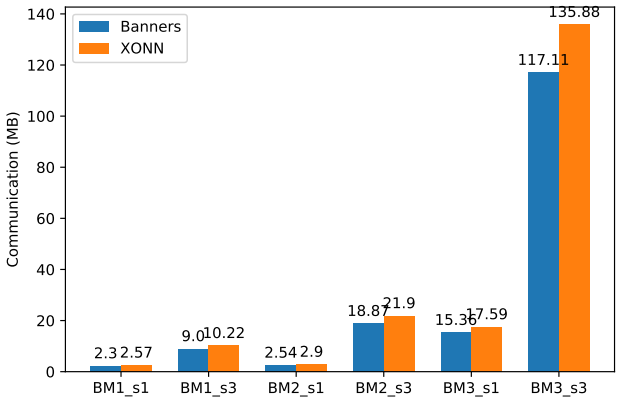

Figure 6: Comparison in communication for MNIST BNN models

All our experiments were ran in a LAN setting on an Intel(R) Core(TM) i7-7800X CPU @ 3.50GHz with 12 cores.

Observing the results from figures 5-10 (detailed results in tables 3 for MNIST models and 4 for CIFAR10 models), we discover that, while the latency is increased around 30%, the communication is slightly lower. We can safely conclude that Banners trades a 30% of speed in exchange for a more robust security model: while XONN offers security against a semihonest adversary, Banners can detect misbehavior and stop the computation.

We can bring the analysis further by comparing all the BNN models in terms of the number of Multiply-Accumulate (MAC) operations. A MAC accounts for an individual VDP operation (elementwise multiplication with cumulative addition), and given that BN and BA are applied elementwise to the output of a VDP, the number of MACs in a model is representative of its complexity[2]. The communication cost is decreased evenly across all models when switching from XONN to Banners, as seen in figures 8 and 12. The latency is higher for all BNN models in our comparison from figures 7 and 11. Furthermore, while the communication increases linearly with the model complexity, there seems to be a certain inherent setup latency: note the almost horizontal slope in figure 8 for the smallest models, affecting both XONN and Banners models. This setup is rendered negligible when the BNN architectures increase in size, such is the case with CIFAR10 models in figure 11.

[2]Note that the direct relation between MACs and complexity applies to sequential NN architectures like the ones described in this paper. It does not hold for Recurrent Neural Networks and other non-sequential NN architectures.

6 CONCLUSION

With the formulation presented in this work, Banners aims to provide an efficient secure inference implementation of BNN, avoiding Oblivious Transfer and other Garbled Circuit primitives by relying on Replicated Secret Sharing. All in all, the memory and space efficiency, coupled with improved security protecting against one malicious adversary, provides a suitable candidate to run secure Biometric Authentication on edge devices.

Future steps of this work will aim at Bit Slicing techniques to obtain considerable parallelization by leveraging on SIMD operations. Additionally, models trained specifically for biometric identification (e.g., face recognition) are envisioned. Last but not least, other libraries besides MP-SPDZ will be targeted.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 265–283.

Table 3: Accuracy, communication, and latency comparisons[2] for MNIST dataset, Banners VS XONN[38].

Arch.	s	# param ($_x10^3$)	# MACs ($_x10^3$)	Accuracy (%)	Communication (MB)		Latency (s)	
					Banners	XONN	Banners	XONN
BM1	1	31	30	97.10	2.30	2.57	0.14	0.12
	1.5	58	57	97.56	3.53	4.09	0.16	0.13
	2	94	93	97.82	5.13	5.87	0.15	0.13
	3	190	188	98.10	9.00	10.22	0.17	0.14
	4	320	316	98.34	13.73	15.62	0.18	0.15
BM2	1	74	91	97.25	2.54	2.90	0.12	0.10
	1.50	153	178	97.93	5.03	5.55	0.14	0.12
	2	291	326	98.28	9.14	10.09	0.16	0.14
	3	652	705	98.56	18.87	21.90	0.21	0.18
	4	1160	1230	98.64	33.42	38.30	0.27	0.23
BM3	1	34	667	98.54	15.36	17.59	0.20	0.17
	1.5	75	1330	98.93	32.22	36.72	0.26	0.22
	2	132	2200	99.13	56.35	62.77	0.36	0.3
	3	293	4610	99.26	117.11	135.88	0.63	0.52
	4	519	7890	99.35	207.40	236.78	0.94	0.81

Table 4: Accuracy, communication, and latency comparisons[2] for CIFAR10 dataset, Banners VS XONN[38].

Arch.	s	# param ($_x10^3$)	# MACs ($_x10^6$)	Accuracy (%)	Communication (GB)		Latency (s)	
					Banners	XONN [3]	Banners	XONN
BC1	1	200	42	0.72	1.22	1.26	5.02	3.96
	1.5	446	92	0.77	2.60	2.82	10.89	8.59
	2	788	163	0.80	4.79	4.98	18.90	15.07
	3	1760	364	0.83	10.31	11.15	43.16	33.49
BC2	1	92	12	0.67	0.37	0.39	1.77	1.37
	1.5	205	27	0.73	0.83	0.86	3.53	2.78
	2	363	47	0.78	1.48	1.53	6.08	4.75
	3	815	105	0.82	3.18	3.40	13.28	10.35
BC3	1	368	41	0.77	1.29	1.35	5.39	4.23
	1.5	824	92	0.81	2.84	3.00	11.52	9.17
	2	1460	164	0.83	4.97	5.32	20.72	16.09
	3	3290	369	0.86	11.03	11.89	45.67	35.77
BC4	1	689	143	0.82	4.36	4.66	17.78	14.12
	1.5	1550	322	0.85	9.88	10.41	39.98	31.33
	2	2750	572	0.87	17.87	18.45	69.36	55.38
	3	6170	1290	0.88	38.56	41.37	158.79	123.94
BC5	1	1210	166	0.81	5.26	5.54	21.17	16.78
	1.5	2710	372	0.85	11.68	12.40	46.78	37.29
	2	4810	661	0.86	20.51	21.98	83.75	65.94
	3	10800	1490	0.88	46.04	49.30	190.14	147.66
BC6	1	1260	23	0.67	0.60	0.65	2.74	2.15
	1.5	2830	50	0.74	1.40	1.46	5.80	4.55
	2	5020	90	0.78	2.48	2.58	10.03	7.91
	3	11300	201	0.80	5.58	5.77	22.44	17.44

[2] The accuracy in Banners models matches the one described in this table by ±0.1%. The number of parameters and number of Multiply-ACcumulate (MAC) are obtained from Larq. The communication and latency for XONN are taken from [38], while figures reported for Banners are yielded by MP-SPDZ.

[3] Although the results in communication of XONN for BC1-BC6 CIFAR10 networks are originally given in MB (see table 12 of [38]), we believe this to be a minor typing error and report them in GB, since BC* models have around 1000 times more MACs than BM* models (compare table 3 with table 4), while the communication costs seem to be lower (table 11 VS table 12 of [38]). This is further confirmed by the mention (appendix A.2 of [38]) of an upper bound of 40GB above which the communication costs are just estimated; which would only make sense if indeed the communication of CIFAR10 models was measured in GB.

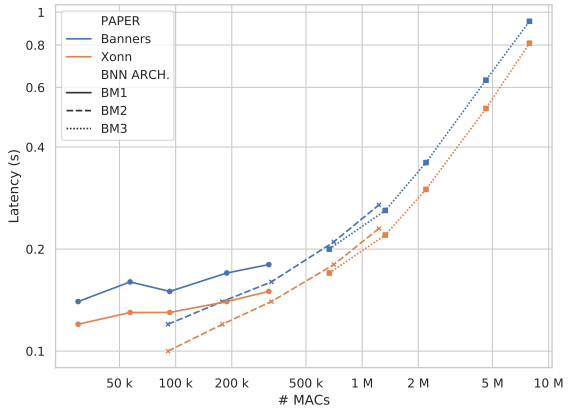

Figure 7: Tradeoff between MACs and Latency for MNIST BNN models

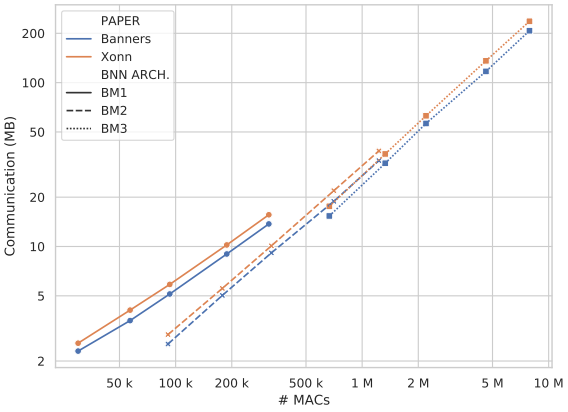

Figure 8: Tradeoff between MACs and Communication for MNIST BNN models

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 308–318.

[3] Anshul Aggarwal, Trevor E Carlson, Reza Shokri, and Shruti Tople. 2020. SOTERIA: In Search of Efficient Neural Networks for Private Inference. *arXiv preprint arXiv:2007.12934* (2020).

[4] Milad Alizadeh, Javier Fernández-Marqués, Nicholas D. Lane, and Yarin Gal. 2019. A Systematic Study of Binary Neural Networks' Optimisation. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rJfUCoR5KX

[5] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 805–817.

[6] Joseph Bethge, Haojin Yang, Marvin Bornstein, and Christoph Meinel. 2019. Back to Simplicity: How to Train Accurate BNNs from Scratch? arXiv:1906.08637 [cs.LG]

[7] Dan Boneh, Amit Sahai, and Brent Waters. 2011. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*. Springer, 253–273.

[8] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. 2018. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*. Springer, 483–512.

[9] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function secret sharing. In *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 337–367.

[10] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[11] Adrian Bulat and Georgios Tzimiropoulos. 2019. XNOR-Net++: Improved binary neural networks. *arXiv preprint arXiv:1909.13863* (2019).

[12] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. 2017. Privacy-Preserving Classification on Deep Neural Network. *IACR Cryptol. ePrint Arch.* 2017 (2017), 35.

[13] Ran Cohen. [n.d.]. Secure Multiparty Computation: Introduction. Open web. https://www.cs.tau.ac.il/~iftachh/Courses/Seminars/MPC/Intro.pdf.

[14] Anders Dalskov, Daniel Escudero, and Marcel Keller. 2020. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies* 2020, 4 (2020), 355–375.

[15] Justin Johnson Fei-Fei Li, Andrej Karpathy. 2016. CNNs in Practice. Open Web slides from lecture. http://cs231n.stanford.edu/slides/2016/winter1516_lecture11.pdf.

[16] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1322–1333.

[17] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. 2017. High-throughput secure three-party computation for malicious adversaries and an honest majority. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 225–255.

[18] Gartner. 2019. Gartner Predictions on Biometric Authentication. *https://www.gartner.com/en/newsroom/press-releases/2019-02-05-gartner-predicts-increased-adoption-of-mobile-centric* (2019).

[19] Lukas Geiger and Plumerai Team. 2020. Larq: An Open-Source Library for Training Binarized Neural Networks. *Journal of Open Source Software* 5, 45 (Jan. 2020), 1746. https://doi.org/10.21105/joss.01746

[20] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 169–178.

[21] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*. 201–210.

[22] Oded Goldreich, Silvio Micali, and Avi Wigderson. 2019. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 307–328.

[23] Koen Helwegen, James Widdicombe, Lukas Geiger, Zechun Liu, Kwang-Ting Cheng, and Roeland Nusselder. 2019. Latent Weights Do Not Exist: Rethinking Binarized Neural Network Optimization. In *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.). Curran Associates, Inc., 7533–7544. http://papers.nips.cc/paper/8971-latent-weights-do-not-exist-rethinking-binarized-neural-network-optimization.pdf

[24] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. 2017. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189* (2017).

[25] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. In *Advances in neural information processing systems*. 4107–4115.

[26] Alberto Ibarrondo and Melek Önen. 2018. FHE-compatible batch normalization for privacy preserving deep learning. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 389–404.

[27] IDEMIA. 2020. Top 4 trends in Biometrics for 2020. *https://www.idemia.com/news/idemias-top-4-trends-biometrics-2020-2020-01-28* (2020).

[28] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *International Conference on Machine Learning*. 448–456.

[29] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1651–1669.

[30] Marcel Keller. 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation. Cryptology ePrint Archive, Report 2020/521. https://eprint.iacr.org/2020/521.

[31] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 336–353.

[32] Baiqiang Liang, Hongrong Ding, Lianfang Huang, Haiqing Luo, and Xiao Zhu. 2020. GWAS in cancer: progress and challenges. *Molecular Genetics and Genomics* (2020), 1–25.
[33] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. 2017. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 619–631.
[34] Payman Mohassel and Peter Rindal. 2018. ABY3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 35–52.
[35] Christine Payne. 2019. MuseNet, 2019. *URL https://openai. com/blog/musenet* (2019).
[36] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*. Springer, 525–542.
[37] Joseph Redmon and Ali Farhadi. 2018. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767* (2018).
[38] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. 2019. XONN: XNOR-Based Oblivious Deep Neural Network Inference. In *Proceedings of the 28th USENIX Conference on Security Symposium*. USENIX Association, USA, 1501–1518.
[39] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*. 707–721.
[40] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* (2014).
[41] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
[42] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[43] Mingxing Tan and Quoc Le. 2019. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In *International Conference on Machine Learning*. 6105–6114.
[44] Florian Tramèr, Fan Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 601–618.
[45] Paulo Vitorino, Sandra Avila, Mauricio Perez, and Anderson Rocha. 2018. Leveraging deep neural networks to fight child pornography in the age of social media. *Journal of Visual Communication and Image Representation* 50 (2018), 303–313.
[46] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2018. SecureNN: Efficient and Private Neural Network Training. *IACR Cryptol. ePrint Arch.* 2018 (2018), 442.
[47] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2020. FALCON: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *arXiv preprint arXiv:2004.02229* (2020).
[48] Wikipedia. 2020. Ripple Carry Adder. Open web. https://en.wikipedia.org/wiki/Adder_(electronics)#Ripple-carry_adder.

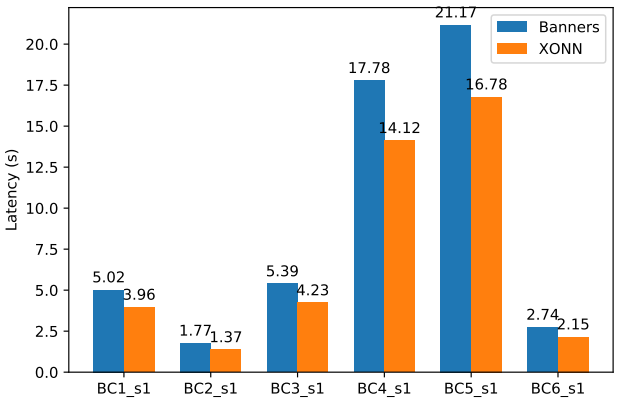

Figure 9: Comparison in latency for CIFAR10 BNN models

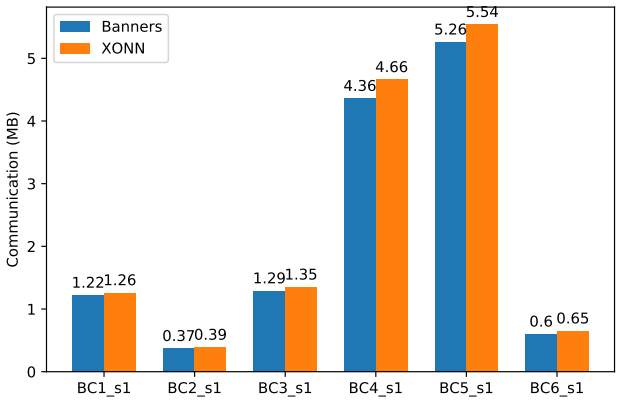

Figure 10: Comparison in communication for CIFAR10 BNN models

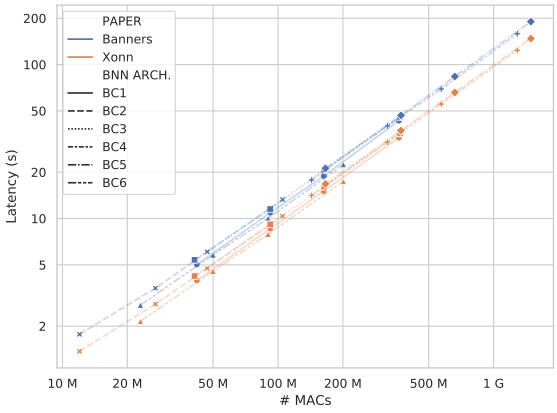

Figure 11: Tradeoff between MACs and Latency for CIFAR10 BNN models

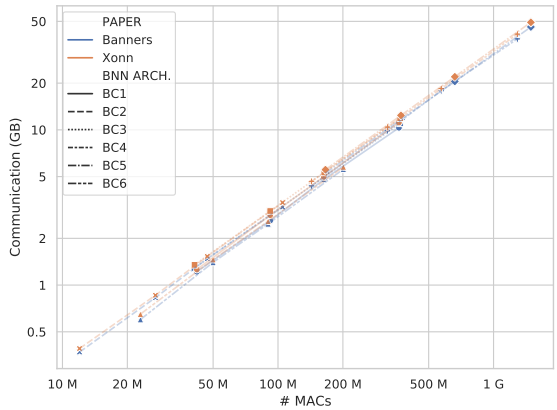

Figure 12: Tradeoff between MACs and Communication for CIFAR10 BNN models

[49] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE, 162–167.
[50] Yang Yu, Zhiqiang Gong, Ping Zhong, and Jiaxin Shan. 2017. Unsupervised representation learning with deep convolutional neural network for remote sensing images. In *International Conference on Image and Graphics*. Springer, 97–108.

APPENDIX

A SECURITY PROOFS

We rely on the existing security proofs of Araki et. al. [5] for the RSS operations (integer multiplication and cumulative addition, XOR, NOT, AND), ABY3[34] (conversion from binary to arithmetic sharing) and FALCON [47] (private compare, as the base for our binary activation) to cover all the primitives in Banners. Further work in these aspects is envisioned.

B GRAPHS FOR CIFAR10 COMPARISON

This appendix holds all the comparison graphs XONN - Banners for CIFAR10 dataset, analogous to those of MNIST dataset present in section 5.